第2章 线性表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

2.1 线性表的类型定义

2.2 线性表的顺序表示和实现

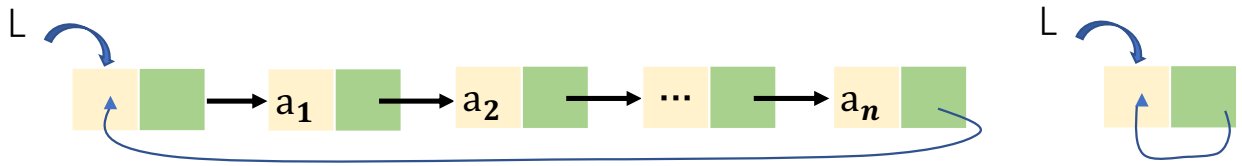
2.3 线性表的链式表示和实现

2.4 顺序表和链表的比较

2.5 线性表的应用

2.3.2 循环链表

- 循环链表：是一种头尾相接的链表（即：表中最后一个结点的指针域指向头节点，整个链表形成一环）。



非空表

空表

优点：从表中任一节点出发均可找到表中其他结点

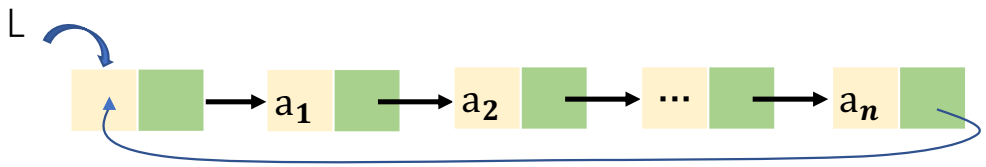
2.3.2 循环链表

注意：由于循环链表中没有NULL指针，故涉及遍历操作时，其终止条件就不再像非循环链表那样判断p或p->next是否为空，而是判断他们是否等于头指针

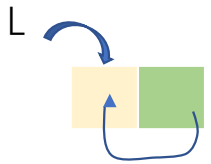
循环条件：

p!=NULL	->	p!=L
p->next!=NULL	->	p->next!=L
单链表		单循环链表

2.3.2 循环链表



非空表



空表

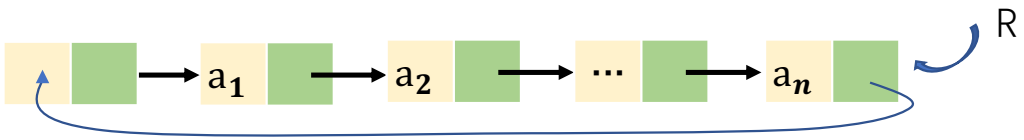
头指针表示单循环链表
- 找a1的时间复杂度 O(1)
- 找an的时间复杂度 O(n)

不方便

注意：表的操作常常是在表的首尾位置上进行

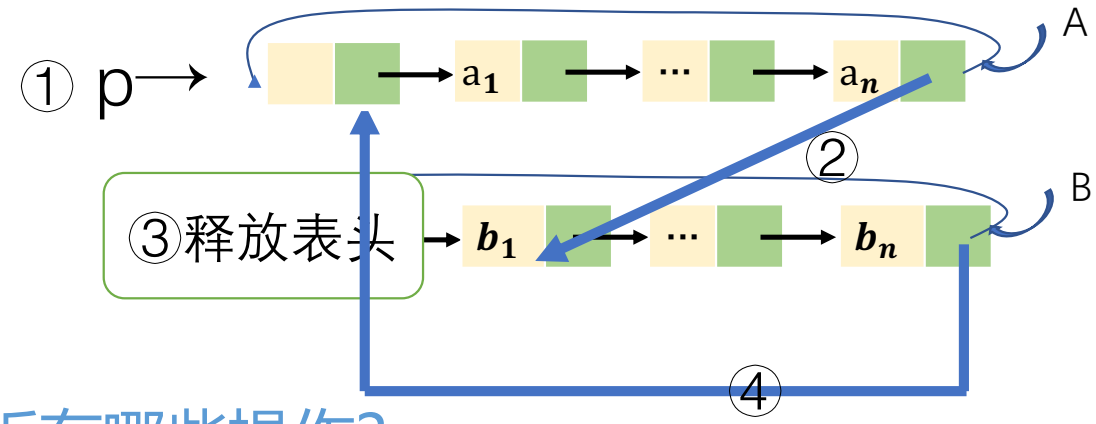
尾指针表示单循环链表
- a1的存储位置是：R->next->next
- an存储位置是：R

时间复杂度 O(1)



2.3.2 循环链表

带尾指针循环链表的合并（将B合并在A之后）



分析有哪些操作?

p存表头结点	B表头连接到A表尾	释放B表头结点	修改指针
p=A->next;	A->next=B->next->next	free (B->next)	B->next=p

2.3.2 循环链表

带尾指针循环链表的合并（将B合并在A之后）

【算法描述】

```
LinkList Connect(LinkList A, LinkList B){
                                        //假设A、B都是非空的单循环链表
    p=A->next;                          //①p存表头结点
    A->next=B->next->next;              //②B表头连接A表尾
    free (B->next);                     //③释放B表头结点
    B->next=p;                          //④修改指针
    return B;
}                                       时间复杂度是O(1)
```

2.3.3 双向链表

为什么要讨论双向链表：

单链表：

单链表的结点->有指示后继的指针域->找后继结点方便
即：查找某结点的后继结点的执行时间为O(1)

->无指示前驱的指针域->找前驱结点难：从表头出发
即：查找某结点的前驱结点的执行时间为O(n)

可用双向链表来克服单链表的这种缺点

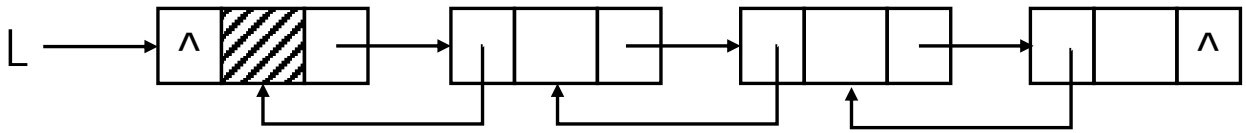
双向链表：在单链表的每个节点里再**增加一个指向其直接前驱的指针**域prior，这样链表中就形成了有两个方向的不同链，故称**双向链表**

2.3.3 双向链表

双向链表的结构可定义如下：

```
typedef struct DuLNode{
  Elemtype          data;
  struct DuLNode   *prior,*next;
}DuLNode, *DuLinkList;
```

（a）结点结构



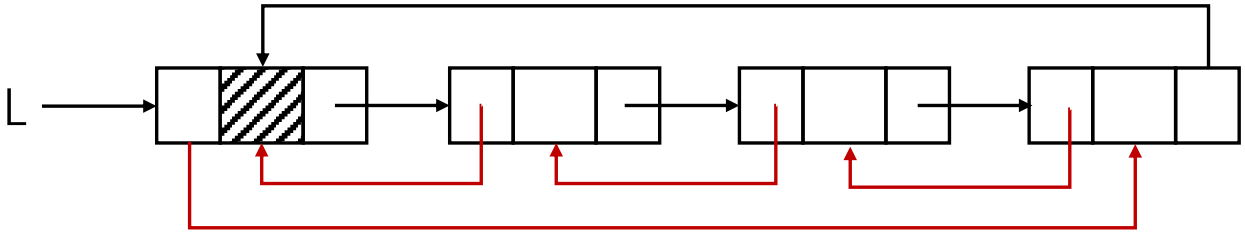
非空表

2.3.3 双向链表

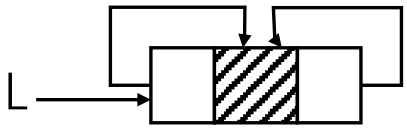
- 双向循环链表

和单链的循环表类似，双向链表也可以有循环表

- 让头结点的前驱指针指向链表的最后一个结点
- 让最后一个结点的后继指针指向头结点



非空的双向循环链表



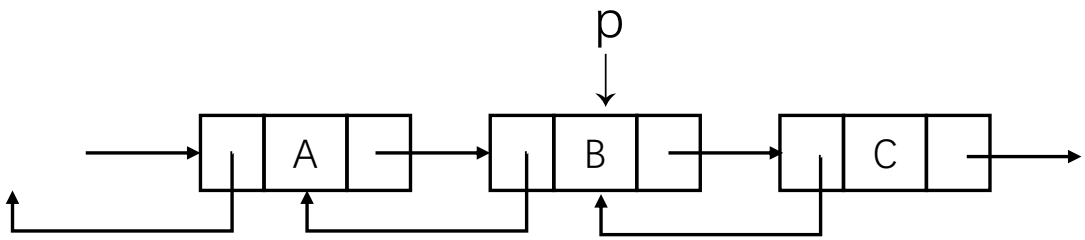
空的双向循环链表

2.3.3 双向链表

双向链表结构的对称性（设指针p指向某一结点）：

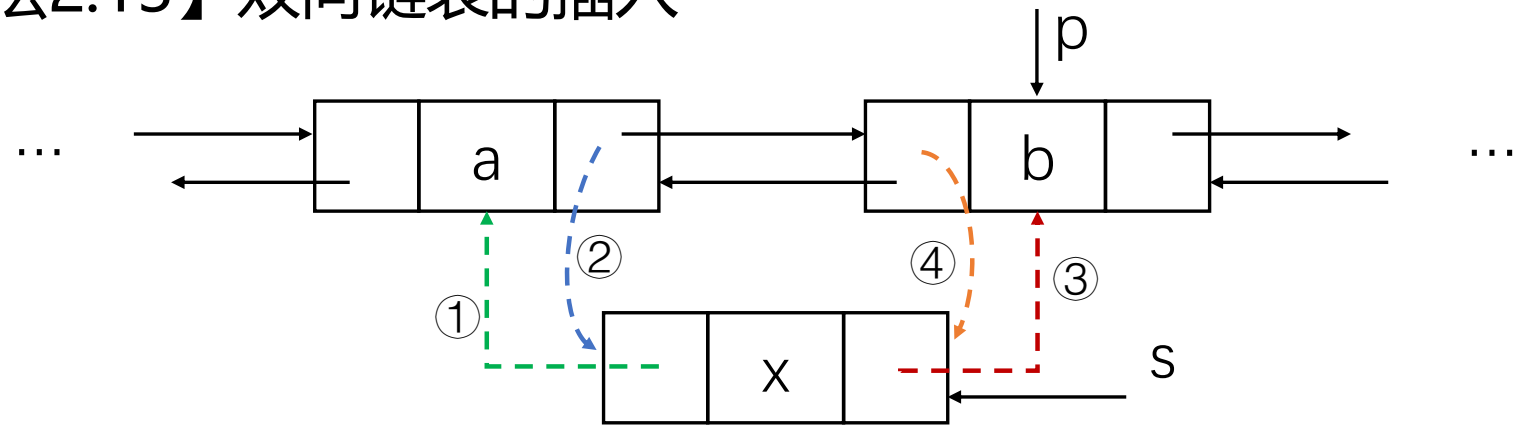
p->prior->next = p = p->next->prior

在双向链表中有些操作（如：ListLength、GetElem等），因仅涉及一个方向的指针，故它们的算法与线性链表的相同。但在插入、删除时，则需同时修改两个方向上的指针。



2.3.3 双向链表

【算法2.13】双向链表的插入



1. s->prior=p->prior;
2. p->prior->next=s;
3. s>next=p;
4. p->prior=s;

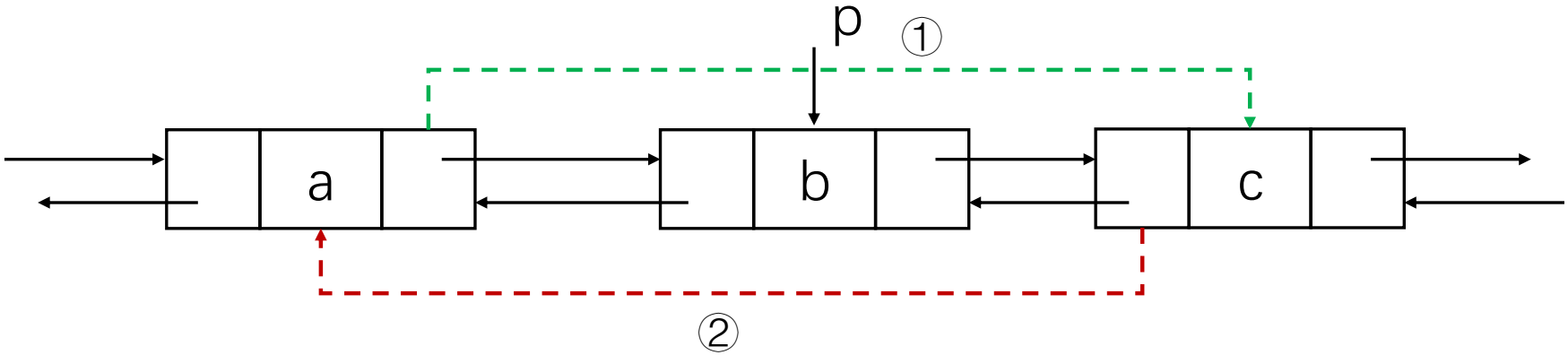
2.3.3 双向链表

【算法2.13】双向链表的插入

```
void LinkInsert_Dul (DuLinkList &L, int i, ElemType e){
//在带头结点的双向循环链表L中第i个位置之前插入元素e
    If(!(p=GetElemP_Dul(L,i))) return ERROR;
     s= (DuLinkList) malloc (sizeof (DuLNode));      s->data=e;
     s->prior=p->prior;      p->prior->next=s;
     s>next=p;               p->prior=s;
     return OK;
} //ListInsert_Dul
```

2.3.3 双向链表

【算法2.14】双向链表的删除



1. p->prior->next=p->next;

2. p->next->prior=p->prior;

2.3.3 双向链表

【算法2.14】双向链表的删除

```
void LinkDelete_Dul (DuLinkList &L, int i, ElemType e){
//删除头结点的双向循环链表L的第i个元素，并用e返回
    If(!(p=GetElemP_Dul(L,i))) return ERROR;
     e=p->data;
     p->prior->next=p->next;
     p->next->prior=p->prior;
     free(p)
     return OK;
} // LinkDelete _Dul
```

单链表、循环链表和双向链表的时间效率比较

	查找表头结点（首元结点）	查找表尾结点	查找结点*p的前驱结点
带头结点的**单链表**L	L->next 时间复杂度O(1)	从L->next依次向后遍历 时间复杂度O(n)	通过p->next无法找到其前驱
带头结点**仅设头指针L**的**循环**单链表	L->next 时间复杂度O(1)	从L->next依次向后遍历 时间复杂度O(n)	通过p->next可以找到其前驱 时间复杂度O(n)
带头结点**仅设尾指针R**的**循环**单链表	R->next->next 时间复杂度O(1)	R 时间复杂度O(1)	通过p->next可以找到其前驱 时间复杂度O(n)
带头结点的**双向循环**链表L	L->next 时间复杂度O(1)	L->prior 时间复杂度O(1)	p->prior 时间复杂度O(1)

第2章 线性表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

2.1 线性表的类型定义

2.2 线性表的顺序表示和实现

2.3 线性表的链式表示和实现

2.4 顺序表和链表的比较

2.5 线性表的应用

2.6 顺序表和链表的比较

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

链式存储结构的优点：

结点空间可以动态申请和释放；

数据元素的逻辑次序靠结点的指针来指示，插入和删除不需要移动数据元素

链式存储结构的缺点：

存储密度小，每个结点的指针域需要额外占用存储空间。当每个结点的数据域所占字节不多时，指针域所占存储空间的比重显得很大。

链式存储结构是非随机存取结构。对任一结点的操作都要从头指针依指针链查到该结点，这增加了算法的复杂度

存储密度

存储密度是指结点数据本身所占的存储量和整个结点结构中所占的存储量之比，即：

$$\text{存储密度} = \frac{\text{数据元素本身占用的存储量}}{\text{结点结构占用的存储量}}$$

例如

存储密度=8/12=67%

一般地，存储密度越大，存储空间的利用率就越高。显然，顺序表的存储密度为1(100%)，而链表的存储密度小于1

2.6 顺序表和链表的比较

比较项目 \ 存储结构		顺序表	链表
空间	存储空间	预先分配，会导致空间闲置或一处现象	动态分配，不会出现存储空间闲置或溢出现象
	存储密度	不用为表示节点间的逻辑关系而增加额外的存储开销，存储密度等于1	需要借助指针来体现元素间的逻辑关系，存储密度小于1
时间	存取元素	随机存取，按位置访问元素的时间复杂度为O(1)	顺序存取，按位置访问元素的时间复杂度为O(n)
	插入、删除	平均移动约一半元素，时间复杂度为O(n)	不许移动元素，确定插入、删除位置后，时间复杂度为O(1)
使用情况		1.表长变化不大，且能事先确定变化的范围 2.很少进行插入或删除操作，经常按元素位置号访问数据元素	1.长度变化较大 2.频繁进行插入或删除操作

第2章 线性表

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

2.1 线性表的类型定义

2.2 线性表的顺序表示和实现

2.3 线性表的链式表示和实现

2.4 顺序表和链表的比较

2.5 线性表的应用

2.5.1 线性表的合并

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

线性表的合并

问题描述：

假设利用两个线性表La和Lb分别表示两个集合A和B，现要求一个新的集合A=AUB

La=(7, 5, 3, 11)　Lb=(2, 6, 3)　→　La=(7, 5, 3, 11, 2, 6)

有序表的合并

问题描述：

已知线性表La和Lb中的数据元素按值非递减有序排列，现要求将La和Lb归并为一个新的线性表Lc，且Lc中的数据元素仍按值非递减有序排列

La=(1, 7, 8)　Lb=(2, 4, 6, 8, 10, 11)　→　Lc=(1, 2, 4, 6, 7, 8, 8, 10, 11)

2.5.1 线性表的合并

线性表的合并

问题描述：

假设利用两个线性表La和Lb分别表示两个集合A和B，现要求一个新的集合A=AUB

La=(7, 5, 3, 11)　Lb=(2, 6, 3)　→　La=(7, 5, 3, 11, 2, 6)

算法步骤：

依次取出Lb中的每个元素，执行以下操作：

1.在La中查找该元素

2.如果找不到，则将其插入La的最后

2.5.1 线性表的合并

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【算法2.15】线性表的合并

```
void union(List &La, List Lb){
        La_len=ListLength(La);
        Lb_len=ListLength(Lb);
        for(i=1; i<=Lb_len; i++){
                GetElem(Lb, i, e);
                if(!LocateElem(La, e)) ListInsert(&La, ++La_len, e)
        }
}
```

算法的时间复杂度是:O(ListLength(La)* ListLength(Lb))

2.5.1 有序表的合并

- 有序表的合并
  - 已知线性表La和Lb中的数据元素按值非递减有序排列，现要求将La和Lb归并为一个新的线性表Lc，且Lc中的数据元素仍按值非递减有序排列

La=(1, 7, 8)　　Lb=(2, 4, 6, 8, 10, 11)　→　　Lc=(1, 2, 4, 6, 7, 8, 8, 10, 11)

算法步骤：

1.创建一个空表Lc

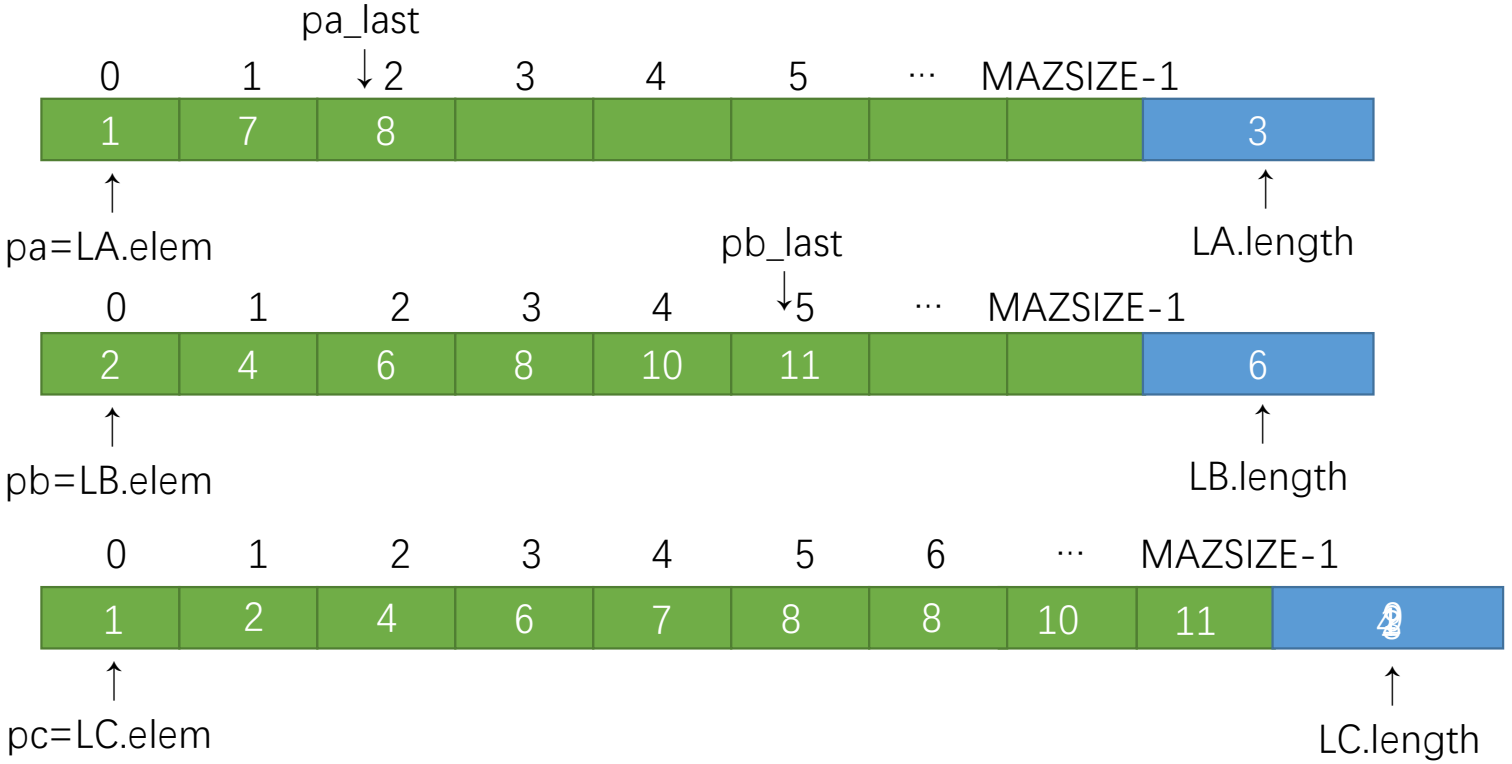
2.依次从La或Lb中“摘取”元素值较小的结点插入到Lc表的最后，直至其中一个表变空为止

3.继续将La或Lb其中一个表的剩余结点插入在Lc表的最后

2.5.1 线性表的合并

杭州电子科技大学 HANGZHOU DIANZI UNIVERSITY

【算法2.16】有序表合并——用顺序表实现



2.5.2 有序表的合并

【算法2.16】有序表的合并——用顺序表实现

```
void MergeList_Sq(SqList LA, SqList LB, SqList &LC){
    pa=LA.elem;
    pb=LB.elem;                    //指针pa和pb初值分别指向两个表的第一个元素
    LC.length=LA.length+ LB.length;   //新表长度为待合并两表的长度之和
    LC.elem =(ElemType*)malloc(sizeof(ElemType)* LC.length);
    //为合并后的新表分配一个数组空间
    pc=LC.elem;                                    //指针pc指向新表的第一个元素
    pa_last=LA.elem+LA.length-1;     //指针pa_last指向LA表的最后一个元素
    pb_last=LB.elem+LB.length-1;     //指针pb_last指向LB表的最后一个元素
```

2.5.2 有序表的合并

【算法2.16】有序表的合并——用顺序表实现

```
//续上页
    while(pa<=pa_last && pb<=pb_last){      //两个表都非空
        if(*pa<=*pb) *pc++=*pa++;  //依次"摘取"两表中值较小的结点
        else *pc++=*pb++;
    }
    while(pa<=pa_last) *pc++=*pa++; //LB表已到达表尾，将LA中剩余元素加入到LC
    while(pb<=pb_last) *pc++=*pb++; //LA表已到达表尾，将LB中剩余元素加入到LC
}//MergeList_Sq
```

```
*pc=*pa;
pc++;
pa++;
```

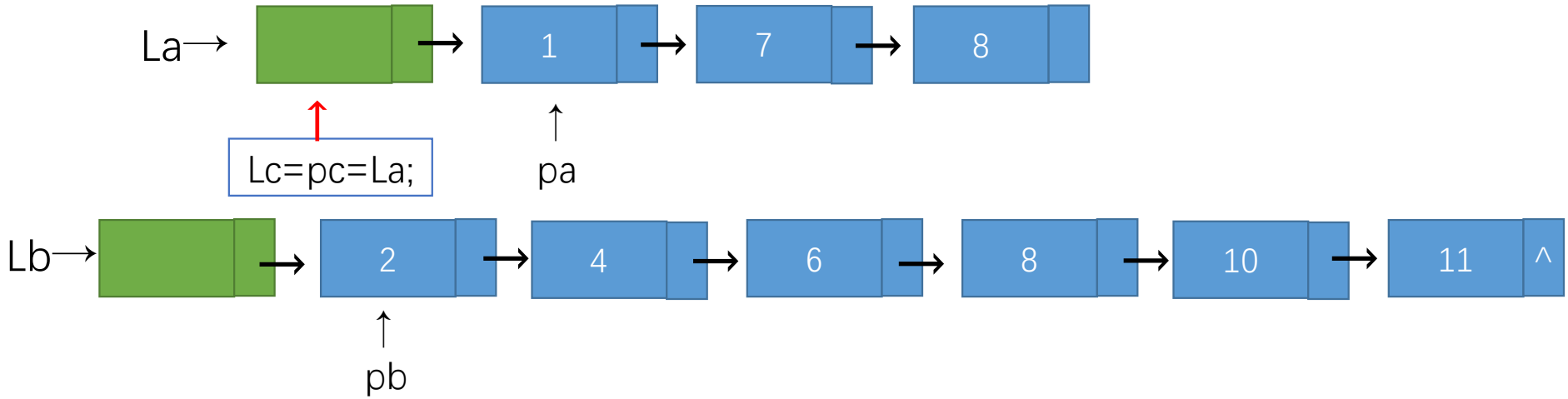
算法的时间复杂度是:O(ListLength(La)+ListLength(Lb))

算法的空间复杂度是:O(ListLength(La)+ListLength(Lb))

2.5.2 有序表的合并

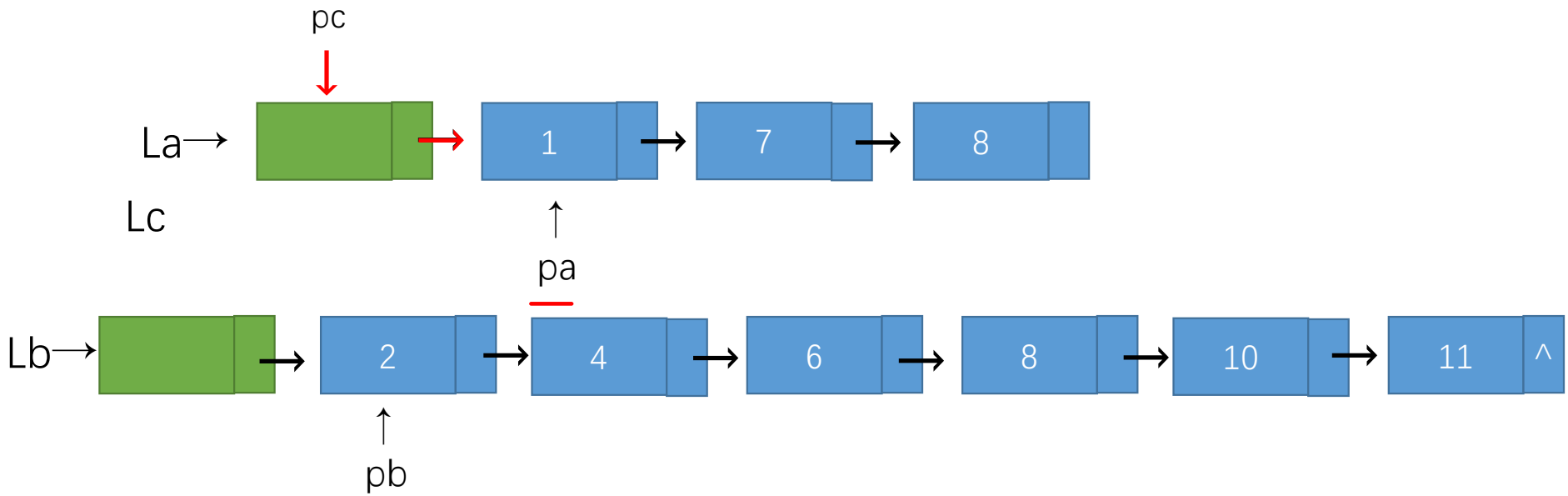
【算法2.17】有序表的合并——用链表实现

用La的头结点作为Lc的头结点



2.5.2 有序表的合并

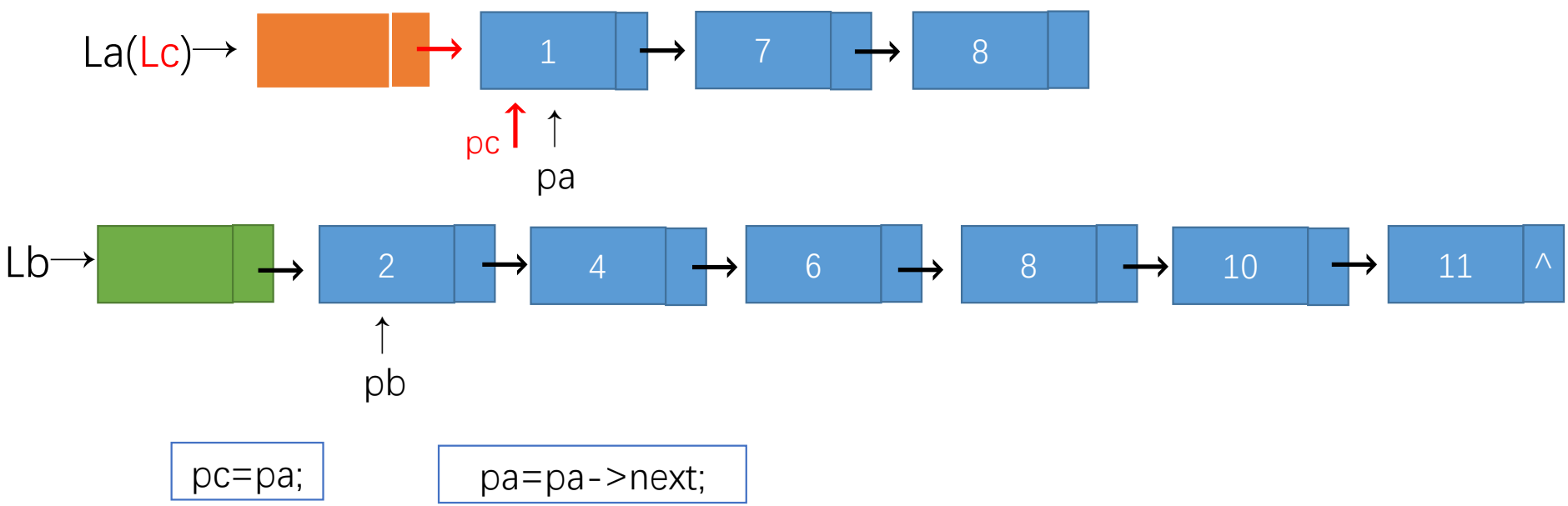
【算法2.17】有序表的合并——用链表实现



pa->data < pb->data → pc->next=pa;

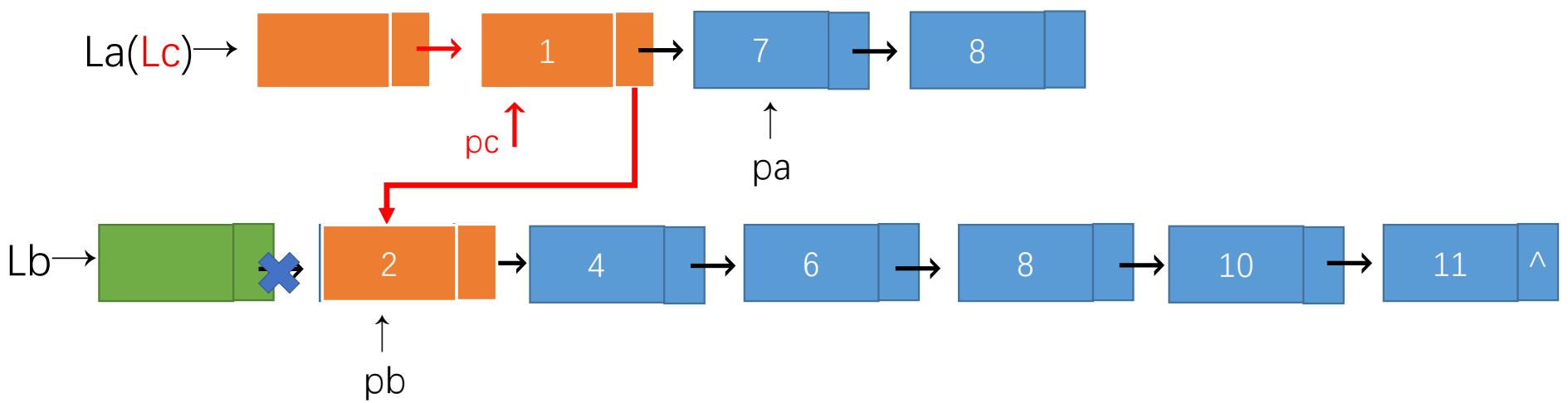
2.5.2 有序表的合并

【算法2.17】有序表的合并——用链表实现



2.5.2 有序表的合并

【算法2.17】有序表的合并——用链表实现



pc->next=pb;

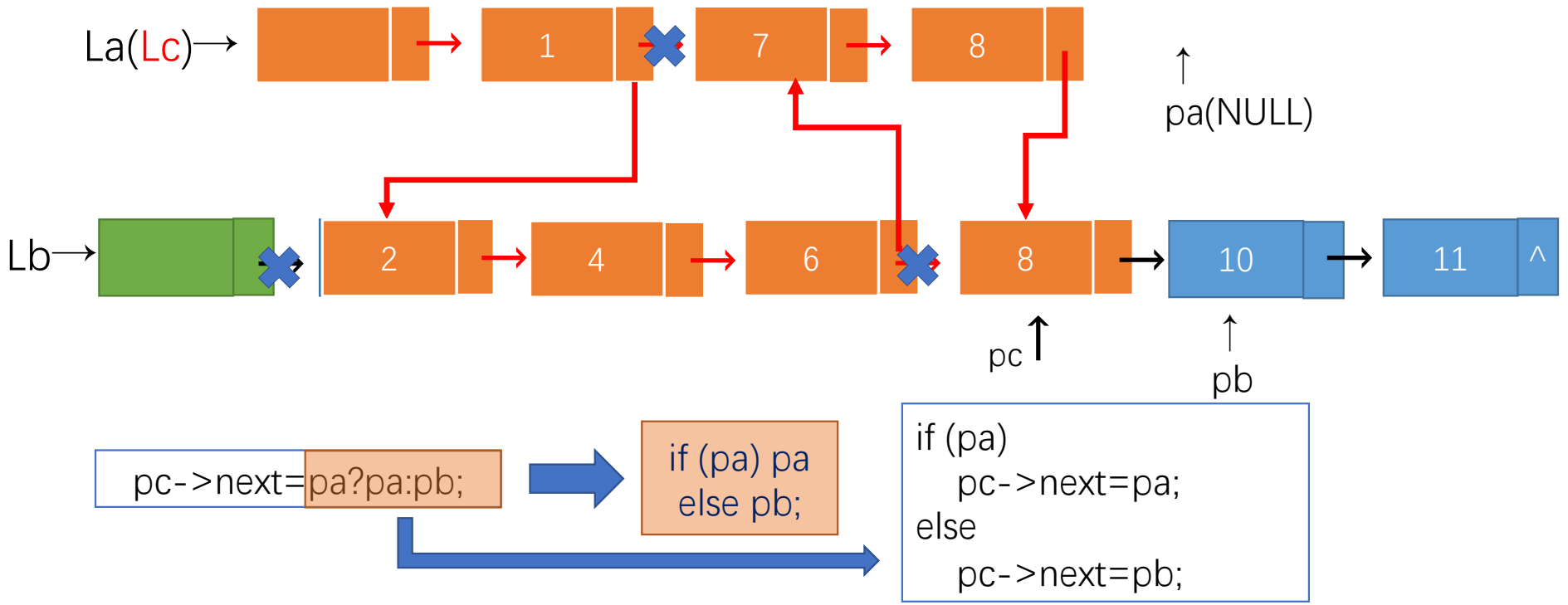
pc=pb;

pb=pb->next;

2.5.2 有序表的合并

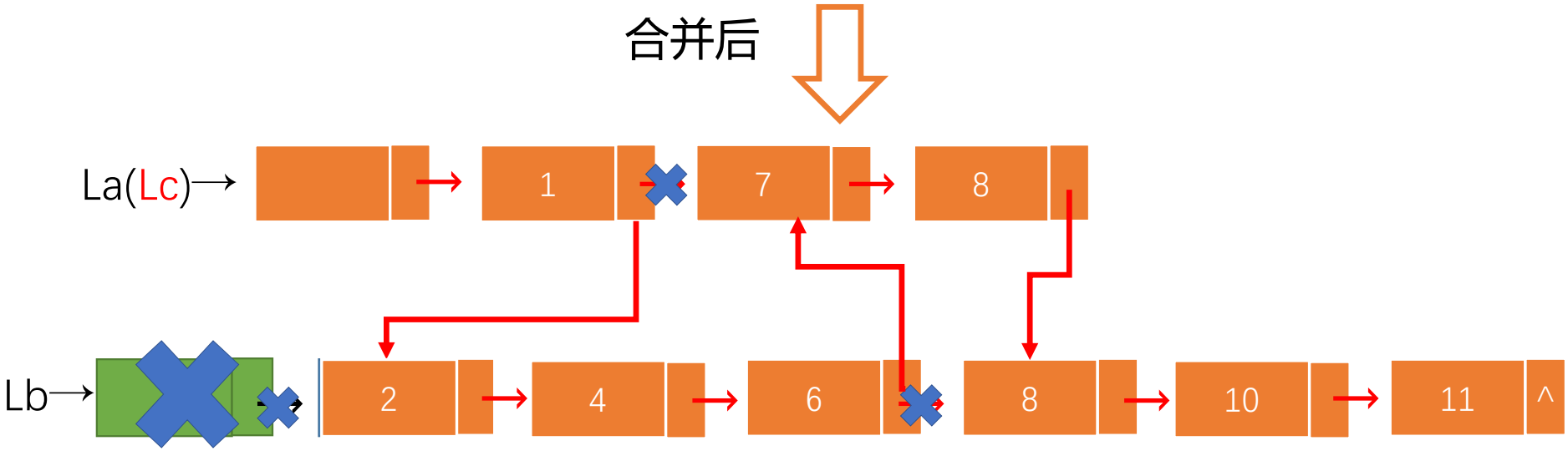
【算法2.17】有序表的合并——用链表实现

插入剩余段



2.5.2 有序表的合并

【算法2.17】有序表的合并——用链表实现



释放Lb的头结点　　free(Lb);

2.5.2 有序表的合并

杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

【算法2.16】有序表的合并——用链表实现

```
void MergeList_L(LinkList &La, LinkList &Lb, LinkList &Lc){
    pa=La->next; pb=Lb->next;
    pc=Lc=La;              //用La的头结点作为Lc的头结点
    while(pa && pb){
        if(pa->data <= pb->data) {pc->next=pa; pc=pa; pa=pa->next;}
        else {pc->next=pb; pc=pb; pb=pb->next}
    }
    pc->next=pa?pa:pb;  //插入剩余段          free(Lb); //释放Lb的头结点
}       算法的时间复杂度是:O(ListLength(La)+ ListLength(Lb))
```